The Ruston BASIC Compiler

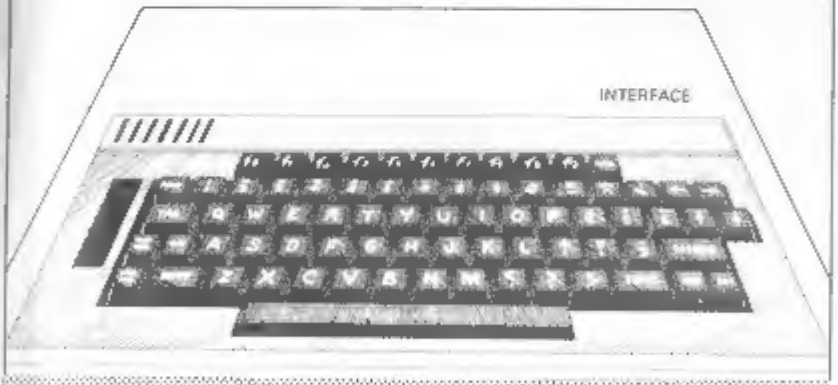
BBC MICRO INSTANT MACHINE CODE

Jeremy Ruston

The Ruston BASIC Compiler

BBC MICRO INSTANT MACHINE CODE

Jeremy Ruston



The Ruston BASIC Compiler

By Jeremy Ruston

The Ruston BASIC Compiler,
first published 1982 in the
United Kingdom by

Interface Publications,
44 - 46 Earls Court Road,
LONDON W8 6EJ.

ISBN 0 907563 18 X

Printed by Short Run Press Ltd, Exeter

COMMERCIAL USE OF PROGRAMS GENERATED BY THE RUSTON BASIC COMPILER

The Publishers welcome the commercial application of the Compiler, but point out that in **all** cases, such application must be discussed prior to release of the software, and a written agreement, incorporating a licence fee and royalty, be completed **before** such software is advertised or distributed in any form, or by any means.

Other publications by Jeremy Ruston available from Interface Publications:

Pascal for Human Beings
The BBC Micro Revealed

Foreword, by Tim Hartnell:

You already know that computers are based on microprocessors. There are very few widely used microprocessors on the market (to all intents and purposes, two - the 6502 (used in the BBC Micro) and the Z80). Although these chips could have been designed to operate with BASIC, they were not. If they were users of any microprocessor would be limited to a single language. So the designers had to make them usable with any language, to ensure maximum sales of their handiwork. They did this by designing the chips to operate with one very simple 'universal' language in which programs could be written. This simple language is called machine code. Because it is so simple, it is called a low level language - other terms you'll hear applied to it are assembly language and machine language. They all mean basically the same thing.

The programs written to allow you to use high level languages like BASIC are called either interpreters or compilers. The difference between the two is largely academic as far as users are concerned. A quite nice analogy to illuminate the differences is that of an American president giving a speech in Norway. The Norwegians will not, of course, be able to understand the president's speech as it stands. Rather than deny them the pleasure, various methods have been devised to overcome the language barrier.

The first, and most obvious, method is to employ a Norwegian who can speak American (or an American who can speak Norwegian) to translate the president's speech as it is made. The second method is to get someone to translate the speech beforehand, and then read out the translated version in tandem with the president. The first method is identical to the action of an interpreter, the second to a compiler. (The third alternative, teaching the president to speak Norwegian, need not be considered). In short, compilers do all the translation before execution and interpreters do it as they go along. This makes compilers a great deal faster.

The Ruston BASIC Compiler is no exception, so I'll hand over to Jeremy Ruston, who will explain it in greater detail.

Tim Hartnell,

London, November 1982.

INTRODUCTION

This manual is divided into two parts. There is a tutorial section occupying the first two thirds and a reference section towards the end of the book. You should read both sections.

You may want to load the Compiler into your computer right now, but to use this book and the program itself efficiently, you should read the tutorial section through carefully, even before you turn the computer on. If you do not do this, you'll find you tend to follow my instructions without really understanding the processes you are carrying out.

The tutorial section is further sub-divided into two sections, each describing one of the ways in which the Compiler can be used. When used in its normal mode, the Compiler reads a program off cassette or disc, compiling it as it goes. Disc users should exclusively use this mode. In the second mode, the Compiler converts a program co-residing in memory with the Compiler itself. The first method is the most powerful, but also the most unwieldy. Cassette users wishing to operate the Compiler in the first mode **must** have equipped their recorders with motor control.

The second method is actually described first, even though it is inferior, since it is easier for the purposes of demonstration.

Earlier versions of the Compiler were shipped with two versions on each cassette - one for disc systems and one for cassette systems. Due to improvements in the design of the Compiler, this is no longer necessary. The single version you have is designed to operate successfully with both cassette - and disc - based BBC Microcomputers.

Chapter One -- Resident mode

This chapter begins the description of the Compiler when operated under the resident mode. As I explained in the introduction, this simply means that both the Compiler and the program to be compiled reside in memory at the same time.

When operated in this mode, it is not possible to use graphics, because the Compiler takes up so much room when combined with the program to be compiled. There simply is no space for any screen mode except MODE 7.

To demonstrate the Compiler in this mode, we will compile this short program:

```
10 TIME=0
20 FOR T%=1 TO 10000
30 NEXT T%
40 PRINT TIME
50 END
```

If you turn on your computer and type in the program in the normal way, you will find that the program runs in about 13.5 seconds. The idea of a compiler is to translate the program into machine code - so it will run much faster.

The next step is to load the Compiler. Cassette users can just type LOAD in the normal way, but disc users should take

this opportunity to transfer the program to disc. This is done with the following sequence of instructions:

```
*TAPE
LOAD "COMPILE"
*DISC
SAVE "COMPILE"
```

The Compiler's cassette also contains a program called RELOC. Do not worry about this for the moment.

The Compiler is about 48 (hex) blocks long, and so takes a long time to load. When you come to use the Compiler on your own programs, you will find that if you use it in the way I shall describe, you shall not have to load it very often.

Once it has loaded, you can LIST it just to make sure. Try to resist the urge to type RUN !

As you may know, there is a special variable called PAGE inside the BBC Computer. This variable governs the place in memory where a program will be loaded and run. The normal value of PAGE is &E00 in cassette systems and &1900 in disc systems.

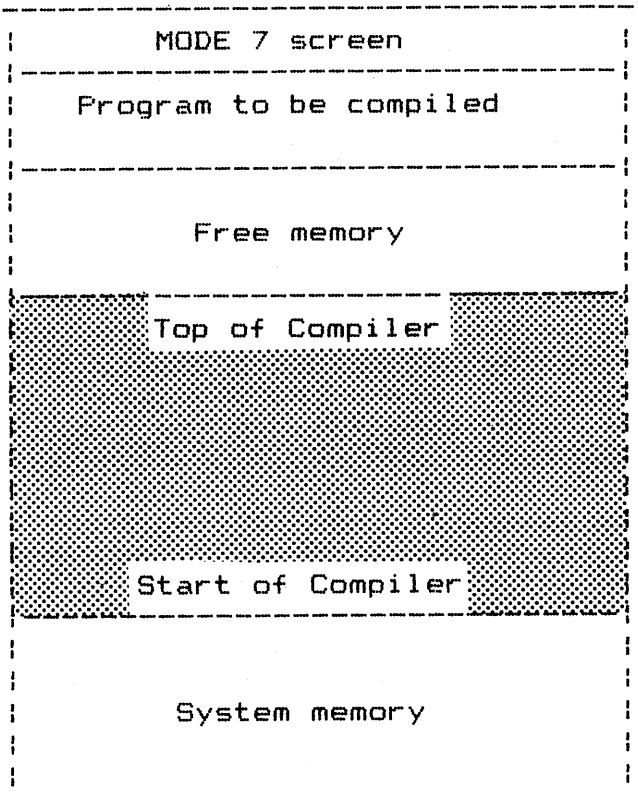
The Compiler extends from PAGE up to a point about 17K above it. The exact point it stops at can be found by typing 'PRINT ~TOP'. Above TOP, the Compiler stores all its variables and arrays.

These will usually last for a couple of K.

Thus there is very little room between the top of the Compilers variables and the bottom of a MODE 7 screen. Into this space we are going to squeeze our short BASIC program, and the machine code generated by it.

The first step is to type in the program to be compiled. We want the computer to store the program in the spare memory between the Compiler and the screen, so we will have to choose a value of PAGE which does not interfere with either. For the moment type 'PAGE=&7800'. Then type NEW in the normal way and proceed to enter the program. Once it has been entered you can run it in the standard way to check it still works.

At this point, the memory of your computer will look something like this:



The diagram is not to scale.

The next step is to ask the Compiler to compile the program and put the machine code generated into the free area of memory outlined above. This machine code can then be executed with the CALL statement as outlined in the User Guide.

To access the Compiler we need to put PAGE back to the start of the Compiler. Thus, owners of cassette systems can type 'PAGE=&E00', and disc owners 'PAGE=&1900'. LIST the program to make sure the compiler is there. (Type OLD if it does not appear).

At last we are ready to RUN the compiler. When you do so, the screen will clear and the message "Is the program to be compiler stored as a file or in memory?" will be displayed. Press the 'M' key. The computer will then fill in the reply as 'in memory'.

The computer will then ask you for 'the PAGE of the source program' ('source' is standard nomenclature for 'the program to be compiled'). Enter '&7800' (the ampersand is vital). The computer will then ask where in memory the machine code should go. Enter '&7000' as this area is free.

The cryptic message 'Press ^B for printer output' will then appear. '^B' is more standard nomenclature, this time meaning 'control-B'. The idea is to give you a chance to press control-B, so that the listing of the program will be sent to a printer, if you have one.

For the moment, press RETURN.

More cryptic messages will then appear.

This time, the computer is printing out the machine code it generated for the program. The machine code is printed in the format of a line of BASIC (preceded by asterisks) followed by the machine code for that particular line. You can examine the machine code at your leisure using control/shift to halt printing.

After all the machine code has been printed, the computer will print some information regarding the length of the code generated.

You are next given the option of saving the code generated to disc/tape. Press 'N' to signify that the code is not to be SAVEd.

The computer will then inform you that CALL N% will start the program. If you follow its advice and type CALL N%, the machine code is executed. You will see that the program now takes about 0.2 seconds to execute.

At the end of this chapter you will find a printout of a sample dialogue with the computer whilst using this mode.

Programs compiled in this way must be short - about 2K or less. If you follow the instructions later in the book, you will discover how to delete bits of the compiler to reduce its length, allowing longer programs to be compiled. The problem is that I have tried to write a

compiler for everyone, but not all users of the program will require the amount of space the Compiler sets aside, for example, for 'sprites', and you could reclaim this and other areas of memory for your own use.

Some general points about the Compiler's operation:

The 'Run-time library' mentioned at the end of the machine code listing is a collection of machine code routines used to carry out various tasks often required in programs. For example, there are routines for all the mathematical operations the Compiler supports and such things as PRINTing and INPUTing a number. This library can be reduced in size with dramatic savings in space.

Compiled programs are generally longer than they were before compilation. Some other compilers for different computers offer shorter compiled programs, but they tend to call many routines inside the BASIC interpreter to do tasks like addition and subtraction. This makes the programs shorter, but means they do not run as fast as programs compiler under the Ruston BBC BASIC Compiler.

The dialect of BASIC compiled was chosen to be a decent - sized subset of BBC BASIC, with the accent on games and other

real-time applications. I therefore make no apologies for the absence of floating point arithmetic.

Not all programs will benefit from compilation. Programs which do a lot of printing are slowed by the printer itself and not the BASIC used. Similarly, graphics programs are not very much faster, but for more subtle reasons. The BBC Micro possesses several routines for drawing lines and plotting points etc. Both my compiler and normal BBC BASIC use these same routines and so are of similar speed in graphics. However, the sprite facility of the Compiler gets around this neatly and efficiently.

The Compiler incorporates quite sophisticated error checking - but never compile a program until it has been thoroughly tested under interpreted BASIC. This is because if a compiled program gets stuck in an endless loop, the only way out is to press 'BREAK'. Make sure all your programs work before you compile them. In this way, the compiler need only be invoked once during the development of a program - cutting down on load times considerably. Note that the Compiler will crash if you face it with a program which jumps out of a FOR loop with a GOTO statement, unlike the interpreter, which just carries on as if nothing had happened.

The Compiler operates slightly different-

ly from the BBC BASIC interpreter in a number of respects. Therefore, in order to maximize the speed of a program, different techniques need to be applied. First, REM statements and spaces do not slow the compiler down at all. Multi-statement lines are not allowed because they would not speed it up at all. Secondly, under BBC BASIC, it is faster to access a variable such as 'A%' than it is to access a constant such as '12'. Under the compiler it is 2/3 faster to use a constant.

Compiled programs should usually be run via a master BASIC program. The BASIC program can do non-time critical things like seting up ENVELOPES and drawing backgrounds. Compiled programs can then do the rest, via a CALL statement in the BASIC program. The ENVELOPE statement is not supported in the compiler because it would take up an awful lot of space whereas under interpreted BASIC it only takes a few bytes.

You should write programs with the compiler in mind, rather than write a program for the interpreter and then say 'Oh well, I might as well compile it now' - if you do write programs with a view to compiling them, you can take advantage of the idiosyncrasies of this and other Compilers to increase the speed of the program beyond that which would be otherwise obtainable.

Here is the version of BASIC the Compiler supports:

[I have included all keywords in this table - those which can only be used in the tape/disc mode are marked with an asterisk]

LET
LET is optional in assignemt statements. LET can assign values to either the variables A% to Z% (the percentage sign may be ommitted under compiled BASIC, but interpreted BASIC prefers it) or the pseudo-variable TIME. The equals sign must be followed by either an expression (see below), another variable (including TIME), or a constant. Further details of these are given towards the end of this chapter.

eg: 'LET A%=A%+1', 'TIME=0'

CLS
CLS is used in the normal way.

eg: 'CLS'

CLG *
CLG operates normally.

eg: 'CLG'

COLOUR *
The COLOUR statement is standard.

eg: 'COLOUR S%', 'COLOUR 128', 'COLOUR G+3'

MODE *
The MODE statement operates normally, but should be avoided. It is better practice to have the master BASIC program change modes for the compiled program.

eg: 'MODE 7', 'MODE 2'

DRAW and **MOVE** *
Both operate normally

eg: 'DRAW X%,Y%', 'DRAW 1000,1000', 'MOVE 3+A%,4*J%'

END
END **must not** be ommitted from the final line of a program. It is possible to follow END with a numerical expression which is then passed to the calling BASIC program via USR. This is described in greater detail later.

eg: 'END'

GCOL *
GCOL is standard.

eg: 'GCOL 0,1', 'GCOL 0,C', 'GCOL X%+1,Y%*2'

GOTO and **GOSUB**
GOTO and GOSUB follow the normal syntax, except that computed destinations are not allowed - ie GOTO/GOSUB must be followed

by a plain number and not an expression such as '1000+T%*100'.

eg: 'GOTO 2340', 'GOSUB 21340'

IF
The IF statement can either omit or include the ELSE clause. In either case, the statements following THEN and ELSE must not be another IF statement. Remembering that GOTO and GOSUB statements are virtually instantaneous, it is good practice to use a GOSUB in an IF statement where you would normally use multistatement lines.

eg: 'IF A%=B% THEN GOTO 200', 'IF A%-2=F% THEN SOUND 0,-15,200,255', 'IF TIME=10 THEN PRINT "A" ELSE PRINT "B"'

OFF
OFF is a non-standard command to turn the cursor off.

eg: 'OFF'

PLOT *
PLOT follows the normal syntax.

eg: 'PLOT 85,1000,1000', 'PLOT 0,-2,-3', 'PLOT 69,X%,Y%'

PRINT
The syntax of the PRINT statement is subtly different from normal. Only three items may be printed: quoted strings, expressions and variables. These may be

punctuated with either a comma, a semi-colon or an apostrophe.
The comma simply prints a space, the semi-colon does nothing and the apostrophe moves to the next line.

eg: 'PRINT 23+A%,"Hello";TIME'

REM
REM can be used without affecting the speed of the machine code. Thus, it makes sense to document your programs more carefully than you would under BBC BASIC. Certain extensions to BBC BASIC are implemented via special characters in REM statements - these are detailed later.

eg: REM 'Evaluate computer's move'

RETURN
Standard.

eg: 'RETURN'

SOUND
The SOUND statement is standard. As detailed in the section on expressions, the SOUND statement is one of the few where a negative constant is meaningful.

eg: 'SOUND 1,-15,B%+4,100'

VDU
The VDU statement follows standard conventions. Both the semi-colon and comma punctuations are allowed - see the

User Guide for more details.

eg: 'VDU 29,640;512;', 'VDU 28,0,20,20,0'

POKE

POKE is implemented in the standard BBC BASIC way - with '?'. The pling operator ('!') is not allowed. The only point worthy of mention is that the expression between the query and the equals sign must not contain an equals sign. (This is most unlikely to occur.)

eg: '?A%=23', '?(A%+N%)=65'

FOR

The FOR statement cannot include a STEP clause, otherwise it is standard.

eg: 'FOR F%=0 TO 1000', 'FOR G%=G% TO T%'

NEXT

The NEXT statement must include a variable name (ie NEXT on its own would be illegal). In addition, only one variable name is allowed for any one NEXT statement (ie NEXT Y,X is illegal).

eg: 'NEXT T%'

REPEAT

The REPEAT statement is standard.

eg: 'REPEAT'

UNTIL

The UNTIL statement is identical to

standard BBC BASIC.

eg: 'UNTIL TIME>1000', 'UNTIL F%=H%'

INPUT

The INPUT statement may only contain one or more variable names, separated by commas. (INPUT TIME is illegal). Thus, quoted strings, apostrophes and the TAB construction are illegal in INPUT statements.

eg: 'INPUT A%', 'INPUT X%,Y%'

***FX**

The FX statement is unusual in that all three parameters must be used. Thus, one must write *FX 15,0,0 rather than the more usual *FX 15.

eg: '*FX 12,A%,C%', '*FX 6,10,0'

CALL

The CALL statement follows standard BBC BASIC conventions, except that in addition to copying the variables A, X and Y into the appropriate processor registers, the variables are updated from the processor registers at the end of the routine. I use the Compiler frequently, and have never had to use this statement, so don't worry if it means little to you.

eg: 'CALL 1000'

Other aspects of the language:

The most important element of a language is the range of expressions it allows. Expressions are simply those parts of a program where you say something like '23+56*(45+A%)'. Expressions are made up of several key elements:

Parenthesis : ()

Numbers : 0-9

Variables : A%-Z% (or A-Z)

Functions : eg ADVAL, INKEY

Operators : eg +, *, -

Functions 'take' a single argument and return a single result - the INKEY statement 'takes' the delay and returns the key pressed.

Operators 'take' two values (whatever is on its left and right) and returns a single value. For example, the sign '+' takes the two things on either side of it and adds them together to get an answer for the sum.

I will now look in detail at these elements.

Numbers:

The Compiler supports numbers in the range 0 to 65535. You can enter negative numbers - but negative numbers are simply added to 65536 - to effectively make them positive. This means that 'PRINT -1' yields 65535. Obviously, this is not much good. Negative numbers are included merely to facilitate use of the SOUND, INKEY and PLOT statements. You can see that adding negative numbers does actually yield true results.

Note - Only positive numbers are recognized by the INPUT statement

(The reason for this curious state of affairs is that (a) it speeds the Compiler up considerably, (b) it allows all memory locations to be accessed (only relevant to experienced programmers) and (c) I consider the more normal range of -32767 to 32767 to be far too limiting.)

Variables:

The Compiler supports 26 integer variables (ie they can only hold whole numbers) called A% to Z%. The percentage sign can be omitted. If you do omit it, the interpreter will treat the variables as floating point ones, and so will run a little slower.

Expressions can also contain the pseudo-variables TIME and GET - both of which act in the same way as BBC BASIC's equivalent statements.

Functions:

The Compiler only supports four functions: INKEY, RND, TAB(X,Y) and ADVAL (POINT is treated as an operator). RND and ADVAL operate in the normal way. So does TAB, which must be used with both arguments and even then only in PRINT statements.

The INKEY statement can either be used with a positive or negative argument. In the former case, it returns the key pressed within a time limit. If no key is pressed it returns -1. With a negative argument, it returns 0 or -1 depending on whether a particular key is pressed down, as detailed in the User Guide. A value of -1 can be tested in a statement like 'IF A=-1 THEN...', even though negative numbers are a little peculiar.

Operators:

The Compiler supplies several new operators. Many of these will only be used by advanced programmers, so details are given later on deleting them, if you decide you can do without some of them.

'+' operates in the normal fashion.

'-' operates normally.

'*' works in the same way as the similar operator in BBC BASIC - except that it

does not report if a multiplication exceeded the limits of the Compiler (came out greater than 65535.)

'[' This operator 'shifts left' the number on its left the number of times indicated by the number on its right. The symbol was chosen because it appears as an arrow on MODE 7. This means that it multiplies the number on the left by two for the number of times indicated by the number on the right. eg, 3[2 is 3*2*2=12. Although this operator has its applications, they are few and far between. You may never use it, but it has been included for completeness.

']' This operator is the opposite of the preceding one, in that it shifts a number right. One use for it is to effectively divide a number by two a specified number of times.

'>' works as in standard BASIC.

'<' works as in standard BASIC.

'£' means 'not equal to', and is thus equivalent to the BASIC operator '<>'. Notice that my printer prints the hash sign as the pound sign, so the above sign is the symbol obtained from shift-3 (#).

'AND' works as in BBC BASIC.

'EOR' works as in BBC BASIC.

'OR' works as in BBC BASIC.

'?' works as in BBC BASIC, except that both arguments must be present. This means that one must write 'O?A%' rather than '?A%'. This is accepted by BBC BASIC, even if it is slightly unorthodox.

'POINT' is implemented in a rather odd way. This is because POINT is the only function taking two arguments. Thus, rather than writing a whole lot of code in the compiler just for this one function, I have implemented it as an operator. All you need to know is that '(X%)READ(Y%)' is the same as 'POINT (X%,Y%)'. This is one of the few features of the compiler which is not the same as BBC BASIC.

The order of precedence of the operators is as follows:

AND
EOR
OR
<
>
£
=
+
-
*
[
]
?
READ

This is the same as the BBC BASIC hierarchy.

You'll see that, with a few exceptions, Compiler BASIC is very close to BBC BASIC.

Now I suggest you try to write a few simple programs in the co-resident mode. Make sure you do not use any of the statements which may only be used in the cassette/disc mode (marked with asterisks in the list above). Try a program of ten lines or less.

When you have proved to yourself just how easy it is to use the Compiler, turn to the next chapter.

```
>REM Compiling in co-resident mode on
 a disc based computer
>PRINT ~PAGE
      1900
>*CAT
Work disc (85)
Drive 0              Option 3 (EXEC)
Directory :0.$       Library :0.$

     !BOOT     L          COMPILE   L
     DNA       L          ENVELOP   L
     FORM40    L          FRENCH    L
     HAL       L          LIMIT     L
     MEMO      L          PASCAL    L
     PUCK      L          RELOC     L
     REM       L          SCRNDMP   L
     SCROLL    L          SPRITE    L
     STAR      L          THREE     L
     TWIST     L          VERIFY    L
>LOAD "COMPILE"
>LIST ,50
   10 REM"----------------------------
   20 REM"BBC BASIC Compiler
   30 REM"(c) Jeremy Ruston
   40 REM"Version 5.2(30 OCT 1982)
   50 REM"----------------------------
>REM So the Compiler has loaded OK
>
>PAGE=&7800
>NEW
>10 TIME=0
>20 FOR T%=1 TO 10000
>30 NEXT T%
>40 PRINT TIME
>50 END
>LIST
   10 TIME=0
   20 FOR T%=1 TO 10000
   30 NEXT T%
   40 PRINT TIME
```

```
   50 END
>REM Run it normally
>RUN
       233
>REM Now return to Compiler
>PAGE=&1900
>L.,50
   10 REM"--------------------------
   20 REM"BBC BASIC Compiler
   30 REM"(c) Jeremy Ruston
   40 REM"Version 5.2(30 OCT 1982)
   50 REM"--------------------------
>REM It is still there !!!!
>RUN


Is the source program in memory (M) or
stored as a file (F) ? in memory


Enter the PAGE of
the source program:&7800


Enter the destination address
for the machine code:&7000

Enter ^B for a printer listing
*** 10 TIME=0
74DC A9 00     lda#(B%MOD256)
74DE 85 2A     staD%
74E0 A9 00     lda#(B%DIV256)
74E2 85 2B     staD%+1
74E4 A2 2A     ldx#&2A
74E6 A0 00     ldy#0
74E8 A9 02     lda#2
74EA 20 F1 FF  jsr&FFF1
*** 20 FOR T%=1 TO 10000
74ED A9 01     lda#(B%MOD256)
```

```
74EF 85 76     staD%
74F1 A9 00     lda#(B%DIV256)
74F3 85 77     staD%+1
*** 30 NEXT T%
74F5 A9 27     lda#(D%DIV256)
74F7 C5 77     cmpB%+1
74F9 D0 06     bneP%+8
74FB A9 10     lda#(D%MOD256)
74FD C5 76     cmpB%
74FF F0 09     beqP%+11
7501 E6 76     incB%
7503 D0 02     bneP%+4
7505 E6 77     incB%+1
7507 4C F5 74  jmpC%
*** 40 PRINT TIME
750A 20 6B 72  jsrtime
750D 20 EF 71  jsrprint
7510 20 E7 FF  jsr&FFE7
*** 50 END
7513 60        rts
7514 60        rts

Start:&7000
End:&7514
(Run-time library ends at &74DB)


Do you want to save the machine code ?N

CALL N% will start the program.
>CALL N%
11
>
>REM It worked
```

Chapter Two - Deleting bits of the Compiler

After the experiments of the last chapter, you may have decided which of the statements/functions you could do without. This chapter describes how to delete parts of the Compiler, to give you more room for programs in the co-resident mode.

The first thing for cassette users is to make a security copy of the Compiler.

Of the two copies you then have, the original should be used when you use the Compiler in cassette/disc mode and the newly - cannibalised one in co-resident mode.

Follow these steps:

1) All users can safely delete the procedure for cassette/disc mode, so type DELETE 6130,6470.

2) You can also safely delete all the parts of the compiler concerned with graphics statements. You will find these at the following line line numbers:

```
DELETE 460,530
DELETE 900,930
DELETE 1180,1270
DELETE 4080,4250
DELETE 6620,6880
```

3) Delete any or all of these ranges as you see fit:
(The symbol/statement on the left is the one affected by deleting the range following it. For example, if you think you can do without addition, type DELETE 590,620.)

INPUT	540,580	
+	590,620	
-	630,660	
PRINT	680,690	
*	700,730	
TIME	740,750	
? (query)	760,780	
AND	790,810	
OR	820,840	
EOR	850,870	
RND	880,890	
READ	900,930	
[	940,970	
]	880,1010	
£	1020,1040	
=	1050,1070	
<	1120,1140	
>	1150,1170	
OFF	3820,3840	
RETURN	3850,3870	
INPUT	3920,3980	
PRINT	4260,4380	
SOUND	4400,4450	
VDU	4460,4500	
POKE	4520,4590	
REPEAT	4610,4620	
UNTIL	4630,4680	
*FX	4690,4730	
GOTO)	4740,5010	GOTO and GOSUB cannot

GOSUB)	be deleted separately.
FOR	5020,5150
NEXT	5160,5290
IF	5480,5640
INKEY	5660,5770
RND	5790,5880
TAB(X,Y)	5900,5980
ADVAL	6000,6110

Where two groups of line numbers are shown in different places for the same statement, delete both ranges.

By carefully deleting some of the above ranges, you will have compacted the compiler a great deal. Once you have done the deletion, do **not** RENUMBER it, as this will make the listing at the end of this manual useless to you.

If you feel you haven't got enough experience of the compiler yet to allow you to decide which bits should be deleted, don't panic, just put off the operation for a few days.

It is important to note down which bits you have deleted, otherwise you may be a little puzzled when the Compiler refuses to compile something simple like 'A%=A%+1' - when you have in fact deleted the plus section!

Having carried out the deletions, SAVE your version of the Compiler. The next stage is to decide on new locations for the machine code and source program in

the new memory you have freed. You will have to use your judgement for this, but it seems sensible to leave at least 2K free for the Compiler's arrays and variables.

Now you should have three copies of the Compiler: the original cassette which came with this manual, your security copy and your newly - cannibalised copy.

As you are safely equipped with a security copy, you can now experiment with the SAVE feature of the compiler.

Compile a program in the usual way. Then, when the computer asks you whether you wish to save the resulting machine code, press Y, to indicate that you do. The computer will then ask you for the filename it should use to do the saving. A *SAVE command is then generated by the compiler and executed. You should see it printed out on the screen. You now have a machine code file on cassette (or disc). The object is to combine this program with a BASIC master program. This is covered under the section on the cassette/disc mode of operation, but for the moment, you can RUN the compiled program by merely typing '*RUN {filename}' . More coverage of this option is given later.

Chapter Three - More esoteric functions

This chapter can be safely skipped by most users. It describes some of the more esoteric features of the compiler - many of which require a good knowledge of both machine code and BBC BASIC.

In the previous chapter I briefly mentioned the ability of the END statement to pass parameters back to the calling BASIC program. This process will now be covered in greater detail.

When BBC BASIC comes across a CALL or USR it takes the values of the variables A,X and Y and copies them into the relevant registers inside the 6502. In the case of the USR function, after the users machine code routine has been executed, the contents of the processors registers are used to build up a value for the USR function. The representation used is:

P, Y, X, A (msb to lsb)

This gives a standard BBC BASIC 32-bit integer.

The END statement can be followed by an expression. The value of the expression is then copied into the processors X and A registers before executing a RTS. As a result, USR(N%) AND &FFFF will execute the compiled program at N% and returns the value after the END statement.

For example, this simple program counts the number of spaces in the MODE 7 screen and returns the number to the calling program:

```
10 A%=0
20 FOR T%=0 TO 999
30 IF T%?31744=32 THEN A%=A%+1
40 NEXT T%
50 END A%
```

Once it has been compiled, the program is access with the line 'PRINT USR(N%) AND &FFFF'.

The program operates by zeroing a counter (A%), then incrementing it at every space found on the screen. This value is then returned in line 50.

If you wish to pass parameters from the calling BASIC program to the Compiled program, simply poke values into the Compiler's variable storage locations:

A	50/51
B	52/53
C	54/55
D	56/57
E	58/59
F	5A/5B
G	5C/5D
H	5E/5F
I	60/61
J	62/63
K	64/65
L	66/67

M	68/69
N	6A/6B
O	6C/6D
P	6E/6F
Q	70/71
R	72/73
S	74/75
T	76/77
U	78/79
V	7A/7B
W	7C/7D
X	7E/7F
Y	80/81
Z	82/83

All addresses are in hexadecimal.

You will notice that for the variable name in A$, these addresses are (ASC(A$)-65)*2+&50 and (ASC(A$)-65)*2+&51.

For example, to use a compiled program to fill a whole MODE 7 screen with a specific character use this code:

Compiled program:

```
10 FOR T%=31744 to 32767
20 ?T%=A%
30 NEXT T%
40 END
```

Then CALL it with:

```
10 INPUT "Enter character code" A%
20 ?&50=A%
30 CALL N%
40 END
```

Chapter Four - Disc / cassette mode

Using this mode of operation is rather more complicated than the previously described method, but it does allow several new commands to be used, as well as making it possible to compile much longer programs.

In this mode, the computer scans through the cassette or disc containing the program three times, each time extracting more information about the program. The program is then compiled in the usual way, except that one must *SAVE the machine code at the end of the compilation sequence. A further program (RELOC, also supplied on your cassette) is then invoked to move the machine code.

The reason why RELOC is needed is not immediately obvious.

The machine code generated by the Compiler is deposited above the program text. This usually means an address of &7000 or so. If that program uses graphics, the graphics screen will overwrite address &7000, so making it impossible to run the program. RELOC simply allows you to alter the starting address of machine code generated by the Compiler.

It is not possible to simply load the program off cassette/disc to a different address because of the way the 6502 microprocessor is designed (the 6502

generates 'non-relocatable' code). Therefore, I have written a special program to carry out the task for you.

The description indicates how time consuming the procedure can be, but if you are careful you will only have to endure it once for each program. Disc users will find this method almost as easy to use as the first.

I should point out that compilers for other computers almost exclusively operate in the second mode, so it is only thanks to the advanced architecture of the BBC Micro that the first mode exists at all.

The program I have chosen for a demonstration uses some of the sprite commands explained in a later chapter - do not worry about them, just type them in.

At the end of this chapter you will find a print out of a typical dialogue with the computer while using this mode. The following comments are to be read in conjunction with that printout.

I started the printer after I had typed in the program, but you can still see the listing.

1) Type in the program, making sure you clear out the Compiler first, by typing NEW.

2) When you have finished, LIST it, as I did, to check it was OK.

3) Save the program to cassette or disc, under the filename DEMO/S. (The '/S' is to indicate that it is a source program).

4) Load the Compiler in the usual way, on top of the program you have just written.

5) RUN it.

6) In answer to the question 'Is the source program stored in memory or as a file ?' answer 'F'.

7) Enter the filename of the program as DEMO/S.

8) Enter the machine code destination address as &7000

9) If you are using a cassette system, you will now have to run the program DEMO/S through three times. Each time the computer prints 'Searching', rewind the DEMO/S cassette to allow it to read the program again.

10) If you have a printer connected to your computer, and would like a listing, enter control-B, followed by RETURN, otherwise, just press the RETURN key.
11) Sit back and watch the machine code.

12) Answer 'Y' to the question 'Do you want to save the machine code?'.

13) The computer will then request the filename under which to save the machine code. Enter "DEMO/O". (The '/O' stands for 'object program', standard nomenclature for a compiled program).

14) Once the Compiler returns you to the command prompt ('>'), type CHAIN "RELOC" to trigger the relocation program.

15) The relocation program will ask you for 'the filename of the source program'. This is a possible source of confusion, since what the RELOC program regards as its source is the file that the Compiler regards as its object file. Enter "DEMO/O".

16) RELOC asks you for the target address. This is the address the machine code will finally occupy. For most programs &2000 is suitable.

17) RELOC then needs the name under which to save the newly - relocated code. I usually use "DEMO/R", for consistency.

18) RELOC will then read DEMO/O. After a pause, it will ask you to press RETURN and record, in order to save the code under DEMO/R. The actual relocation process is quite long.

19) Once this has been completed, you can load and run the program, as demonstrated in the printout.

Now, any process which requires 19 steps to carry out is potentially liable to errors, so if the above procedure does not work, try again, because I promise you that it does work ! Once you understand the above process you may wish to try it out on a few programs. . (A point to watch for disc users is that the Compiler uses a routine in the BASIC ROM for generating random numbers. If you're programs involve the function RND, they must not be invoked with *RUN, but with *LOAD and then CALL. This is to enable the BASIC ROM to replace the DFS ROM. This is only applicable to disc users, and is a minor point, but it can cause unusual problems if you are not prepared for it.

That is a good cue for an explanation of how to merge BASIC and compiled programs.

The technique is to RELOCate the program to &2000, then write a BASIC calling program, making sure it does not go up to &2000 (check this with 'PRINT ~TOP'). Then save the BASIC program on tape or disc, making its last line is *RUN (program name). Then you can run the whole lot with a simple CHAIN "".

```
>REM Compiling in disc/cassette mode
>
>REM I have a program in memory:
>LIST
   10 MODE 4
   20 OFF
   30 REM SPRITE 0,170,85,170,85,170,85,
170,85
   40 REPEAT
   50 FOR T%=8 TO 248
   60 REM WAIT
   70 REM DRAW 0,T%,T%
   80 REM WAIT
   90 REM DRAW 0,T%,T%
  100 NEXT T%
  110 UNTIL INKEY(-1)
  120 END
>SAVE "DEMO/S"
>LOAD "COMPILE"
>RUN


Is the source program in memory (M) or
stored as a file (F) ? in a file



Enter the filename of
the source program:DEMO/S


Enter the destination address
for the machine code:&7000

Enter ^B for a printer listing
*** 10 MODE 4
74DC A9 16     lda#T%
74DE 20 EE FF  jsrwr
74E1 A9 04     lda#(B%MOD256)
74E3 20 EE FF  jsrwr
```

```
*** 20 OFF
74E6 A9 0A     lda#10
74E8 8D 00 FE sta&FE00
74EB A9 20     lda#32
74ED 8D 01 FE sta&FE01
*** 30 REM SPRITE 0,170,85,170,85,170,85
,170,85
*** 40 REPEAT
*** 50 FOR T%=8 TO 248
74F0 A9 08     lda#(B%MOD256)
74F2 85 76     staD%
74F4 A9 00     lda#(B%DIV256)
74F6 85 77     staD%+1
*** 60 REM WAIT
74F8 78        sei
74F9 AD 4D FE .lkj lda&FE4D
74FC 29 02     and#2
74FE F0 F9     beqlkj
7500 A9 FF     lda#255
7502 8D 4D FE sta&FE4D
7505 58        cli
*** 70 REM DRAW 0,T%,T%
7506 A6 76     ldxB%
7508 A4 76     ldyB%
750A 4C 10 75 jmpP%+6
7510 AD 0E 75 ldaP%-2
7513 85 8D     sta&8D
7515 AD 0F 75 ldaP%-6
7518 85 8E     sta&8E
751A 20 6D 74 jsrsprite
*** 80 REM WAIT
751D 78        sei
751E AD 4D FE .lkj lda&FE4D
7521 29 02     and#2
7523 F0 F9     beqlkj
7525 A9 FF     lda#255
7527 8D 4D FE sta&FE4D
752A 58        cli
*** 90 REM DRAW 0,T%,T%
752B A6 76     ldxB%
```

```
752D A4 76     ldyB%
752F 4C 35 75  jmpP%+6
7535 AD 33 75  ldaP%-2
7538 85 8D     sta&8D
753A AD 34 75  ldaP%-6
753D 85 8E     sta&8E
753F 20 6D 74  jsrsprite
*** 100 NEXT T%
7542 A9 F8     lda#D%
7544 C5 76     cmpB%
7546 F0 05     beqP%+7
7548 E6 76     incB%
754A 4C F8 74  jmpC%
*** 110 UNTIL INKEY(-1)
754D A9 7C     lda#124
754F 20 F4 FF  jsr&FFF4
7552 A2 FF     ldx#(C%MOD256)
7554 A0 FF     ldy#(C%DIV256)
7556 A9 81     lda#129
7558 20 F4 FF  jsr&FFF4
755B 98        tya
755C 48        pha
755D 8A        txa
755E 48        pha
755F 68        pla
7560 AA        tax
7561 68        pla
7562 8A        txa
7563 C9 00     cmp#0
7565 D0 03     bneP%+5
7567 4C F0 74  jmpR%(RE%)
*** 120 END
756A 60        rts
756B 60        rts

Start:&7000
End:&756B
(Run-time library ends at &74DB)
```

```
Do you want to save the machine code ?Y

Enter the filename : DEMO/O
*SAVE DEMO/O 7000+56B 7040
Successful save

CALL N% will start the program.
>REM Do not type CAlL N%!
>CHAIN "RELOC"
Enter the filename
of the source program:DEMO/O

Enter the target address:&2000

Enter the filename
of the object file:DEMO/R
*SAVE DEMO/R 7000+56B 2040 2000
>REM Now try it...
>
>*RUN DEMO/R
>REM It worked...
```

Chapter Five - Sprites

This chapter describes the extensions to BBC BASIC provided by the Compiler.

These new statements only work in MODE 4. They are accessed through special REM statements.

Sprites are 'objects' defined in an 8 by 8 grid - exactly like the user - defined graphics you are used to. The difference is that sprites may be moved around the screen without disturbing what is on the screen already. Sprites can also safely pass in front of, and behind, each other.

The new statements are:

```
REM SPRITE
REM DRAW
REM UP
REM DOWN
REM LEFT
REM RIGHT
REM WAIT
```

They work like this:

REM SPRITE defines the shape of a sprite, in the same way as VDU 23 defines characters. Sprites are numbered 0 to 7.

A typical sprite definition could be:

```
REM SPRITE 0,170,85,170,85,170,85,170,85
```

This defines sprite 0 to be a chequered square. The relevant bytes are filled up inside the run-time library. The definition must not involve variables - just literal constants.

REM DRAW A,X,Y draws sprite number A at position X,Y. A must also be a number, not an expression. X and Y can be anything. The exact operation of the statement is to exchange the shape at X,Y with the definition. Thus, whatever was on the screen behind the sprite is saved. To remove the sprite, redraw the same sprite at the same position.

This makes for very fast graphics, but has the side effect that a particular sprite may only be at one place at any one time.

REM UP, REM LEFT, REM UP and REM DOWN simply scroll the screen one direction in any direction.

REM WAIT waits for single vertical refresh pulse. The effect is that by putting REM WAITs into a program you can ensure that the sprites do not flicker.

The program we compiled in the last chapter illustrate most of these statements.

There is one disadvantage with these statements. The right hand 'quarter' of the screen cannot be used. The coord-

inates used for sprites extend from 0 to 255 in both directions, with the origin at the top left.

If you have more than one sprite on the screen at the same time, there are some points to remember. So that the sprites can pass over each other safely, you must draw and redraw them in a different order. If you consider the process of drawing the sprites carefully you will see why this is so.

Using the sprite facility is surprisingly easy except for one small disadvantage - an error in any program you may write using sprites will cost cassette users some 10 minutes in loading and saving. If you read through a program carefully and dry run it in your head before running, you will avoid many errors and save a lot of amount of time.

Before you launch into long sessions of experimenting with sprites, it makes sense to ensure that you can compile normal programs efficiently and without error.

Chapter Six - Reference

This chapter describes the statements and functions available on the Compiler in the same format as the BASIC keywords are defined in the User Guide.

[LET] <variable name>=<expression>

CLS

CLG

COLOUR <expression>

MODE <expression>

DRAW <expression>,<expression>

MOVE <expression> , <expression>

END [<expression>]

GCOL <expression> , <expression>

GOTO <num-const>

GOSUB <num-const>

IF <testable condition> THEN <statement> [ELSE <statement>]

OFF

PLOT <expression> , <expression> , <expression>

```
PRINT  { <expression> | <string-const>  ,
| ' | <null> }

REM

REM  SPRITE <num-const> ,  <num-const>  ,
<num-const> , <num-const> , <num-const> ,
<num-const> , <num-const> , <num-const> ,
<num-const>

REM  DRAW <num-const>  ,  <expression>  ,
<expression>

REM UP

REM DOWN

REM LEFT

REM RIGHT

REM WAIT

RETURN

SOUND  <expression>  ,  <expression>  ,
<expression> , <expression>

VDU <expression> {,|;<numeric>}[;]

? <expression> = <expression>

FOR  <variable  name> =  <expression>  TO
<expression>

NEXT <variable name>
```

REPEAT

UNTIL <testable condition>

INPUT <variable name> {, <variable name>}

*FX <expression> , <expression> , <expression>

CALL <expression>

```
>LIST
   10 REM"----------------------------
   20 REM"BBC BASIC Compiler
   30 REM"(c) Jeremy Ruston
   40 REM"Version 5.2(30 OCT 1982)
   50 REM"----------------------------
   60
   70
   80
   90
  100MODE7
  110DIMLL%(50,1),R%(15),FO%(15,1):LC%=1
:RE%=1:FR%=1:MC$="Missing ,"
  120PRINT''"Is the source program in me
mory (M) or  stored as a file (F) ? ";:A
$=GET$:IFA$<>"F"ANDA$<>"f"ANDA$<>"m"ANDA
$<>"M"GOTO120
  130IFA$="F"ORA$="f"THENFL%=FALSE:PRINT
"in a file"ELSEFL%=TRUE:PRINT"in memory"
  140IFFL%GOTO170
  150INPUT'''"Enter the filename of"'"th
e source program:"FI$
  160GOTO190
  170INPUT'''"Enter the PAGE of"'"the so
urce program:"A$
  180D%=EVAL(A$)
  190INPUT''"Enter the destination addre
ss"'"for the machine code:"A$
  200M%=EVAL(A$)
  210N%=M%+64
  220sp=M%
  230PROClibrary(M%+64)
  240Q%=P%
  250INPUT'"Enter ^B for a printer listi
ng"A$
  260IFFL%PROCcompileM ELSEPROCcompileF
  270PRINT'"Start:&";~M%'"End:&";~P%-1'"
(Run-time library ends at &";~Q%-1;")"
  280K%=P%-1
  290PROCsecondpass
```

```
  300VDU3
  310PRINT''"Do you want to save the mac
hine code ?";:REPEATA$=GET$:UNTILA$="Y"O
RA$="y"ORA$="N"ORA$="n"
  320PRINTA$:IFA$="Y"ORA$="y"PROCsave
  330PRINT'"CALL N% will start the progr
am."
  340END
  350
  360DEFPROClibrary(M%)
  370LOCALT%
  380FORT%=0TO2STEP2
  390P%=M%
  400[OPTT%
  410jmp libend
  420.string
  430pla:sta&84:pla:sta&85:.again clc:ld
a#1:adc&84:sta&84:lda#0:adc&85:sta&85:ld
y#0
  440lda(&84),Y:cmp#13:beqexit:jsr&FFEE:
jmpagain:.exit lda&85:pha
  450lda&84:pha:rts
  460.down
  470lda#0:sta&8F:.nextx lda#0:sta&90:st
a&84:.nexty jsraddress:ldy#0:lda(&86),Y:
sta&85:lda&84:sta(&86),Y:lda&85:sta&84:i
nc&90:bnenexty:clc:lda#8:adc&8F:sta&8F:b
nenextx
  480.up
  490 lda#0:sta&8F:.nxtx lda#255:sta&90:
lda#0:sta&84:.nxty jsraddress:ldy#0:lda(
&86),Y:sta&85:lda&84:sta(&86),Y:lda&85:s
ta&84:dec&90:bnenxty:clc:lda#8:adc&8F:st
a&8F:bnenxtx:rts
  500.leftscl
  510 php:lda#0:sta&90:.nixty plp:clc:ph
p:lda#255:sta&8F:.nixtx jsraddress:ldy#0
:lda(&86),Y:plp:rolA:sta(&86),Y:php:sec:
lda&8F:sbc#8:sta&8F:cmp#255:bnenixtx:inc
&90:bnenixty:plp:rts
```

```
  520.rightscl
  530 php:lda#0:sta&90:.naxty plp:clc:ph
p:lda#0:sta&8F:.naxtx jsraddress:ldy#0:l
da(&86),Y:plp:rorA:sta(&86),Y:php:clc:ld
a&8F:adc#8:sta&8F:bnenaxtx:inc&90:bnenax
ty:plp:rts
  540.input
  550pla:sta&84:pla:sta&85:pla:sta&86:ld
a#0:sta&87:.sdf ldy#0:.get lda#124:jsr&F
FF4:jsr&FFE0:cmp#127:beqdel:sta&600,Y:cm
p#13:beqcr:jsr&FFEE:iny:jmpget:.cr tya:b
eqsdf:jsr&FFE7:lda#0:sta&88:sta&89:ldy#0
  560.annie lda&88:sta&8A:lda&89:sta&8B:
asl&8A:rol&8B:ldx#3:.rte clc:rol&88:rol&
89:dex:bnerte:clc:lda&8A:adc&88:sta&88:l
da&8B:adc&89:sta&89
  5701da&600,Y:sec:sbc#&30:clc:adc&88:st
a&88:lda#0:adc&89:sta&89:iny:lda&600,Y:c
mp#13:bneannie
  5801da&88:ldy#0:sta(&86),Y:lda&89:iny:
sta(&86),Y:lda&85:pha:lda&84:pha:rts:.de
l tya:beqget:dey:lda#127:jsr&FFEE:jmpget
  590.add
  600pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:sta&89
  610clc:lda&86:adc&88:sta&86:lda&87:adc
&89:pha:lda&86:pha
  6201da&85:pha:lda&84:pha:rts
  630.subtract
  640pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:sta&89
  650sec:lda&88:sbc&86:sta&86:lda&89:sbc
&87:pha:lda&86:pha
  6601da&85:pha:lda&84:pha:rts
  670.print
  680pla:sta&84:pla:sta&85:pla:sta&87:pl
a:sta&88:ldy#0:.sbd2 ldx#16:lda#0:.sbd3
asl&87:rol&88:rolA:cmp#10:bccsbd4:sbc#10
:inc&87:.sbd4 dex
  690bnesbd3:pha:iny:lda&88:ora&87:bnesb
```

```
d2:.sbd5 pla:clc:adc #&30:jsr&FFEE:dey:b
nesbd5:lda&85:pha:lda&84:pha:rts
  700.multiply
  710pla:sta&8A:pla:sta&8B:pla:sta&84:pl
a:sta&85:pla:sta&86:pla:sta&87:lda#0:sta
&88:sta&89
  7201dx#16:.leep asl&88:rol&89:asl&86:r
ol&87:bccnadd:lda&84:clc:adc&88:sta&88
  7301da&85:adc&89:sta&89:.nadd dex:bnel
eep:lda&89:pha:lda&88:pha:lda&8B:pha:lda
&8A:pha:rts
  740.time
  750pla:sta&84:pla:sta&85:ldx#&2A:ldy#0
:lda#1:jsr&FFF1:lda&2B:pha:lda&2A:pha:ld
a&85:pha:lda&84:pha:rts
  760.query
  770pla:sta&8E:pla:sta&8F:jsradd:pla:st
a&84:pla:sta&85
  7801da#0:pha:ldy#0:lda(&84),Y:pha:lda&
8F:pha:lda&8E:pha:rts
  790.and
  800pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:and&87
  810pha:lda&88:and&86:pha:lda&85:pha:ld
a&84:pha:rts
  820.or
  830pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:ora&87
  840pha:lda&88:ora&86:pha:lda&85:pha:ld
a&84:pha:rts
  850.eor
  860pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:eor&87
  870pha:lda&88:eor&86:pha:lda&85:pha:ld
a&84:pha:rts
  880.rnd
  890pla:sta&8A:pla:sta&8B:pla:sta&2A:pl
a:sta&2B:lda#0:sta&2C:sta&2D:jsr&AF41:ld
a&2B:pha:lda&2A:pha:lda&8B:pha:lda&8A:ph
a:rts
```

```
  900.point
  910pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:sta&89
  9201da#9:ldy#0:ldx#&86:jsr&FFF1:lda#0:
pha:lda&8A:pha:lda&85:pha
  9301da&84:pha:rts
  940.left
  950pla:sta&84:pla:sta&85:pla:sta&87:pl
a:pla:sta&88:pla:sta&89
  960.shft lda&87:cmp#0:beqnoshft:clc:as
l&88:rol&89:dec&87:jmpshft
  970.noshft lda&89:pha:lda&88:pha:lda &
85:pha:lda&84:pha:rts
  980.right
  990pla:sta&84:pla:sta&85:pla:sta&87:pl
a:pla:sta&88:pla:sta&89
 1000.rshft lda&87:cmp#0:beqnorshft:clc:
lsr&89:ror&88:dec&87:jmprshft
 1010.norshft lda&89:pha:lda&88:pha:lda&
85:pha:lda&84:pha:rts
 1020.neql
 1030pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla
 1040cmp&87:bnetrue:lda&88:cmp&86:bnetru
e:beqfalse
 1050.eql
 1060pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla
 1070eor&87:bnefalse:lda&88:eor&86:bnefa
lse:beqtrue
 1080.true
 10901da#&FF:pha:pha:lda&85:pha:lda&84:p
ha:rts
 1100.false
 11101da#&0:pha:pha:lda&85:pha:lda&84:ph
a:rts
 1120.lss
 1130pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:sta&89
 1140sec:lda&86:sbc&88:lda&87:sbc&89:bcs
```

```
true:bccfalse
 1150.grt
 1160pla:sta&84:pla:sta&85:pla:sta&86:pl
a:sta&87:pla:sta&88:pla:sta&89
 1170sec:lda&86:sbc&88:lda&87:sbc&89:bcs
false:bcctrue
 1180.address lda&8F:and#&F8:sta&86:lda&
90:and#7:ora&86:sta&86
 11901da&90:and#&18:aslA:aslA:aslA:clc:a
dc&86:sta&86:lda#0:adc#&58
 1200sta&87:lda&90:and#&E0:lsrA:lsrA:lsr
A:lsrA:lsrA:clc:adc&87:sta&87
 12101da&90:and#&F8:lsrA:lsrA:lsrA:clc:a
dc&87:sta&87:rts
 1220.sprite stx&8F:sty&90:lda#0:sta&91:
.bnm
 1230jsraddress:lda&8F:and#7:tax:ldy#0:l
da(&86),Y:sta&84:ldy#8:lda(&86),Y:sta&85
:.ery txa:beq ext:asl&85:rol&84:dex:bnee
ry:.ext
 12401dy&91:lda(&8D),Y:sta&88:lda&84:sta
(&8D),Y
 12501da#255:sta&8B:lda#0:sta&8A:sta&89:
jsraddress:lda&8F:and#7:tay:.lopr tya:be
q nFtrot:lsr&88:ror&89:sec:ror&8A:ror&8B
:dey:jmplopr
 1260.nFtrot lda(&86),Y:and&8A:ora&88:st
a(&86),Y:ldy#8:lda(&86),Y:and&8B:ora&89:
sta(&86),Y
 1270inc&90:inc&91:lda&91:cmp#8:bnebnm:r
ts
 1280.libend
 1290]NEXT:ENDPROC
 1300
 1310
 1320DEFFNNOSPC(A$)
 1330LOCALB$,C$,T%
 1340FORT%=1TOLEN(A$)
 1350C$=MID$(A$,T%,1)
 1360IFC$=CHR$(34)GOTO1400
```

```
1370IFC$<>" "ANDC$<>"%" B$=B$+C$
1380NEXT
1390=B$
1400B$=B$+C$:T%=T%+1:C$=MID$(A$,T%,1)
1410IFC$=CHR$(34)GOTO1370
1420IFT%>LEN(A$)PROCERR("Missing """)
1430GOTO1400
1440
1450DEFFNPULL
1460LOCALA$
1470IFLEN(S$)=0:=""
1480A$=LEFT$(S$,1)
1490S$=MID$(S$,2)
1500=A$
1510
1520DEFPROCPUSH(A$)
1530S$=A$+S$
1540ENDPROC
1550
1560DEFFNNEXT
1570LOCALB$
1580IFA%>LEN(A$):=""
1590B$=MID$(A$,A%,1)
1600A%=A%+1
1610=B$
1620
1630DEFFNCUR
1640IFA%>LEN(A$):="" ELSE=MID$(A$,A%,1)
1650
1660DEFFNTOP
1670IFLEN(S$)=0:="" ELSE=LEFT$(S$,1)
1680
1690DEFFNNUM
1700LOCALB$,T%
1710IFFNCUR="+"ORFNCUR="-"GOTO1790
1720IFFNCUR>"9"ORFNCUR<"0":=-1
1730REPEAT
1740B$=B$+FNNEXT
1750UNTILFNCUR>"9"ORFNCUR<"0"
1760T%=VAL(B$)
```

```
 1770IFSGN(T%)=-1 T%=T%+65536
 1780=T%
 1790B$=B$+FNNEXT
 1800IFFNCUR>"9"ORFNCUR<"0"PROCERR("Miss
ing number")
 1810GOTO1730
 1820
 1830DEFFNVAR
 1840IFFNCUR>"Z"ORFNCUR<"A":=-1
 1850=&50+(ASC(FNNEXT)-65)*2
 1860
 1870DEFFNPREC(A$)=INSTR("("+CHR$(&80)+C
HR$(&82)+CHR$(&84)+")><#=+-*[]"+CHR$(&3F)
+CHR$(&F3),A$)
 1880
 1890DEFPROCEXP(A$)
 1900LOCALS$,B$,B%,C$,F$,A%
 1910S$="":A%=1
 1920IFFNCUR="("THENPROCPUSH(FNNEXT):GOT
O1920
 1930B%=FNNUM
 1940IFB%<>-1[lda#(B%DIV256):pha:lda#(B%
MOD256):pha:]GOTO2040
 1950B%=FNVAR
 1960IFB%<>-1[ldaB%+1:pha:ldaB%:pha:]GOT
O2040
 1970B$=FNNEXT
 1980IFB$=CHR$(&A5)[lda#124:jsr&FFF4:lda
#0:pha:jsr&FFE0:pha:]GOTO2040
 1990IFB$=CHR$(&B3)PROCrnd:GOTO2040
 2000IFB$=CHR$(&A6)PROCinkey:GOTO2040
 2010IFB$=CHR$(&96)PROCadval:GOTO2040
 2020IFB$=CHR$(&91)[jsrtime:]GOTO2040
 2030PROCERR("Missing operand")
 2040IFFNCUR=")"GOTO2170
 2050IFFNPREC(FNCUR)=0PROCERR("Missing o
perator")
 2060B$=FNNEXT
 2070IFB$=""GOTO2130
 2080C$=FNTOP
```

```
 2090IFC$=""PROCPUSH(B$):GOTO1920
 2100IFFNPREC(C$)<FNPREC(B$)PROCPUSH(B$)
:GOTO1920
 2110PROCRATE(FNPULL)
 2120GOTO2080
 2130REPEAT
 2140IFFNTOP<>""PROCRATE(FNPULL)
 2150UNTILFNTOP=""
 2160ENDPROC
 2170F$=FNNEXT
 2180B$=FNPULL
 2190IFB$=""PROCERR("Unmatched brackets"
)
 2200IFB$="("GOTO2040
 2210PROCRATE(B$)
 2220GOTO2180
 2230IFSGN(B%)>-1[lda#(B&DIV256):pha:lda
#(B%MOD256):pha:]GOTO2040 ELSE B%=B%+655
36:[lda#(B%DIV256):pha:lda#(B%MOD256):ph
a:]GOTO2040
 2240
 2250DEFPROCRATE(V$)
 2260IFV$="+"[jsradd:]ENDPROC
 2270IFV$="-"[jsrsubtract:]ENDPROC
 2280IFV$="*"[jsrmultiply:]ENDPROC
 2290IFV$="["[jsrleft:]ENDPROC
 2300IFV$="]"[jsrright:]ENDPROC
 2310IFV$=">"[jsrgrt:]ENDPROC
 2320IFV$="<"[jsrlss:]ENDPROC
 2330IFV$="="[jsreql:]ENDPROC
 2340IFV$="#"[jsrneql:]ENDPROC
 2350IFV$=CHR$(&80)[jsrand:]ENDPROC
 2360IFV$=CHR$(&82)[jsreor:]ENDPROC
 2370IFV$=CHR$(&84)[jsror:]ENDPROC
 2380IFV$=CHR$(&F3)[jsrpoint:]ENDPROC
 2390IFV$=CHR$(&3F)[jsrquery:]ENDPROC
 2400PROCERR("Badly matched brackets")
 2410
 2420DEFPROCcompileM
 2430LOCALAD%,wr
```

```
2440AD%=D%
2450L%=0:wr=&FFEE
2460REPEAT
2470L%=L%+1
2480AD%=AD%+AD%?3
2490UNTILAD%?1=255
2500DIMLI%(L%,1)
2510L%=0
2520AD%=D%
2530REPEAT
2540L%=L%+1
2550LI%(L%,0)=?(AD%+1)*256+AD%?2
2560AD%=AD%+AD%?3
2570UNTILAD%?1=255
2580L%=0
2590REPEAT
2600L%=L%+1
2610LI%(L%,1)=P%
2620PROCoutput($(D%+4))
2630PROCstatement($(D%+4))
2640D%=D%+D%?3
2650UNTILD%?1=255
2660[rts:]
2670ENDPROC
2680
2690DEFPROCoutput(T$)
2700LOCALCH%,A%
2710PRINT;~*** ";LI%(L%,0);
2720FORCH%=1TOLEN(T$)
2730A%=ASC(MID$(T$,CH%,1))
2740CALL&B53A
2750NEXT
2760PRINT
2770ENDPROC
2780
2790DEFPROCstatement(A$)
2800LOCALA%,SD$
2810A%=1
2820A$=FNNOSPC(A$)
2830SD$=FNNEXT
```

```
 2840IFSD$=CHR$(&E9)PROClet(FNNEXT)
 2850IFSD$=CHR$(&D6)PROCcall
 2860IFSD$=CHR$(&DA)PROCcl(16)
 2870IFSD$=CHR$(&DB)PROCcl(12)
 2880IFSD$=CHR$(&FB)PROCcolmod(17)
 2890IFSD$=CHR$(&DF)PROCdrmo(5)
 2900IFSD$=CHR$(&E0)PROCend
 2910IFSD$=CHR$(&E6)PROCgcol
 2920IFSD$=CHR$(&E4)PROCtosub(FALSE)
 2930IFSD$=CHR$(&E5)PROCtosub(TRUE)
 2940IFSD$=CHR$(&E7)PROCif
 2950IFSD$=CHR$(&EB)PROCcolmod(22)
 2960IFSD$=CHR$(&EC)PROCdrmo(4)
 2970IFSD$=CHR$(&87)PROCoff
 2980IFSD$=CHR$(&F0)PROCplot
 2990IFSD$=CHR$(&F1)PROCprint
 3000IFSD$=CHR$(&F4)PROCrem
 3010IFSD$=CHR$(&F8)PROCreturn
 3020IFSD$=CHR$(&D4)PROCsound
 3030IFSD$=CHR$(&EF)PROCvdu
 3040IFSD$=CHR$(&3F)PROCpoke
 3050IFSD$=CHR$(&E3)PROCfor
 3060IFSD$=CHR$(&ED)PROCnext
 3070IFSD$=CHR$(&F5)PROCrepeat
 3080IFSD$=CHR$(&FD)PROCuntil
 3090IFSD$=CHR$(&E8)PROCinput
 3100IFSD$="*"ANDMID$(A$,A%,2)="FX"PROCf
x
 3110IF(SD$>="A"ANDSD$<="Z")ORSD$=CHR$(&
D1)PROClet(SD$)
 3120IFSD$<>""PROCERR("UNKNOWN STATEMENT
")
 3130ENDPROC
 3140
 3150DEFPROCERR(Z$)
 3160G%=0:D%=A%:PRINT':FORT%=1TOLEN(A$):
A%=ASC(MID$(A$,T%)):CALL&B53A:IFT%=D%:G%
=COUNT-2
 3170NEXT:IFD%>=T% G%=T%
 3180PRINT'TAB(G%);"^"'CHR$(3)'"ERROR --
```

```
";Z$
3190CLOSE#0
3200END
3210
3220DEFFNV(A$)
3230IFLEN(A$)=1ANDA$<="Z"ANDA$>="A":=&5
0+(ASC(A$)-65)*2ELSE=-1
3240
3250DEFFNN(A$)
3260LOCALT%,C$
3270T%=1
3280IFLEFT$(A$,1)="-":T%=T%+1
3290C$=MID$(A$,T%,1)
3300IFC$>"9"ORC$<"0":=-1
3310T%=T%+1
3320IFT%<=LEN(A$)GOTO3290
3330=(VAL(A$)+65536)MOD65536
3340
3350DEFFNITEM
3360LOCALB$,C$:B$=FNCUR:IFB$="'"ORB$=";
"ORB$=","ORB$=CHR$(&B8)ORB$=CHR$(&88):=F
NNEXT
3370C$="":REPEATB$=FNNEXT:C$=C$+B$
3380IFA%<=LEN(A$):B$=FNCUR ELSEB$=","
3390UNTILB$=","ORB$="'"ORB$=";"ORB$=CHR
$(&B8)ORB$=CHR$(&88)
3400=C$
3410
3420DEFFNITEN
3430LOCALB$,C$:B$=FNCUR:IFB$=CHR$(&8C)O
RB$=CHR$(&8B):=FNNEXT
3440C$="":REPEATB$=FNNEXT:C$=C$+B$
3450IFA%<=LEN(A$):B$=FNCUR ELSEB$=CHR$(
&8C)
3460UNTILB$=CHR$(&8B)ORB$=CHR$(&8C)
3470=C$
3480
3490DEFPROCcl(T%)
3500SD$=""
3510[lda#T%:jsrwr:]ENDPROC
```

```
 3520
 3530DEFPROCVDBYT(B$)
 3540LOCALB%:B%=FNN(B$):IFB%<>-1[lda#(B%
MOD256):jsrwr:]ENDPROC
 3550B%=FNV(B$):IFB%<>-1[ldaB%:jsrwr:]EN
DPROC
 3560PROCEXP(B$):[pla:jsrwr:pla:]ENDPROC
 3570
 3580DEFPROCVDWRD(B$)
 3590LOCALB%:B%=FNN(B$):IFB%<>-1[lda#(B%
MOD256):jsrwr:lda#(B%DIV256):jsrwr:]ENDP
ROC
 3600B%=FNV(B$):IFB%<>-1[ldaB%:jsrwr:lda
B%+1:jsrwr:]ENDPROC
 3610PROCEXP(B$):[pla:jsrwr:pla:jsrwr:]E
NDPROC
 3620
 3630DEFPROCSTBYT(B$,D%)
 3640LOCALB%:B%=FNN(B$):IFB%<>-1[lda#(B%
MOD256):staD%:]ENDPROC
 3650B%=FNV(B$):IFB%<>-1[ldaB%:staD%:]EN
DPROC
 3660PROCEXP(B$):[pla:staD%:pla:]ENDPROC
 3670
 3680DEFPROCSTWRD(B$,D%)
 3690LOCALB%:B%=FNN(B$):IFB%<>-1[lda#(B%
MOD256):staD%:lda#(B%DIV256):staD%+1:]EN
DPROC
 3700B%=FNV(B$):IFB%<>-1[ldaB%:staD%:lda
B%+1:staD%+1:]ENDPROC
 3710PROCEXP(B$):[pla:staD%:pla:staD%+1:
]ENDPROC
 3720DEFPROCrem
 3730SD$="":LOCALB$:B$=MID$(A$,A%):IFB$=
"WAIT"[sei:.lkj lda&FE4D:and#2:beqlkj:ld
a#255:sta&FE4D:cli:]ENDPROC
 3740IFLEFT$(B$,6)="SPRITE"PROCSPRITE(MI
D$(B$,7)):ENDPROC
 3750IFLEFT$(B$,4)="DRAW"PROCDRAW(MID$(B
$,5)):ENDPROC
```

```
 3760IFB$="UP"[jsrup:]ENDPROC
 3770IFB$="DOWN"[jsrdown:]ENDPROC
 3780IFB$="LEFT"[jsrleftscl:]ENDPROC
 3790IFB$="RIGHT"[jsrrightscl:]ENDPROC
 3800ENDPROC
 3810
 3820DEFPROCoff
 3830SD$="":[lda#10:sta&FE00:lda#32:sta&
FE01:]ENDPROC
 3840
 3850DEFPROCreturn
 3860SD$="":[rts:]ENDPROC
 3870
 3880DEFPROCend
 3890SD$="":LOCALB$:B$=FNITEM:IFB$=""[rt
s:]ENDPROC
 3900PROCEXP(B$):[pla:tay:pla:tax:tya:rt
s:]ENDPROC
 3910
 3920DEFPROCinput
 3930SD$="":LOCALB$
 3940B$=FNNEXT:IFB$>"Z"ORB$<"A"PROCERR("
Bad variable name")
 3950[lda#(&50+(ASC(B$)-65)*2):pha:jsrin
put:]
 3960B$=FNNEXT:IFB$=""ENDPROC
 3970IFB$=","GOTO3940
 3980PROCERR("Illegal INPUT syntax")
 3990
 4000DEFPROClet(D$)
 4010SD$="":IF(D$>"Z"ORD$<"A")ANDD$<>CHR
$(&D1)PROCERR("Bad variable name")
 4020IFFNNEXT<>"="PROCERR("Missing = sig
n")
 4030LOCALC%:C%=&50+(ASC(D$)-65)*2:IFD$=
CHR$(&D1) C%=&2A
 4040PROCSTWRD(MID$(A$,A%),C%)
 4050IFD$<>CHR$(&D1)ENDPROC
 4060[ldx#&2A:ldy#0:lda#2:jsr&FFF1:]ENDP
ROC
```

```
 4070
 4080DEFPROCcolmod(T%)
 4090SD$="":LOCALB$:[lda#T%:jsrwr:]B$=MI
D$(A$,A%):PROCVDBYT(B$):ENDPROC
 4100DEFPROCcall
 4110SD$="":PROCSTWRD(MID$(A$,A%),&85):[
lda#&4C:sta&84:ldx&7E:ldy&80:lda&50:jsr&
84:sta&50:txa:sta&7E:tya:sta&80:lda#0:st
a&51:sta&7F:sta&81:]ENDPROC
 4120
 4130DEFPROCdrmo(T%)
 4140SD$="":[lda#25:jsrwr:lda#T%:jsrwr:]
PROCVDWRD(FNITEM):IFFNNEXT<>",PROCERR(M
C$)
 4150PROCVDWRD(FNITEM):ENDPROC
 4160
 4170DEFPROCgcol
 4180SD$="":[lda#18:jsrwr:]PROCVDBYT(FNI
TEM):IFFNNEXT<>",PROCERR(MC$)
 4190PROCVDBYT(FNITEM):ENDPROC
 4200
 4210DEFPROCplot
 4220SD$="":[lda#25:jsrwr:]PROCVDBYT(FNI
TEM):IFFNNEXT<>",PROCERR(MC$)
 4230PROCVDWRD(FNITEM):IFFNNEXT<>",PROC
ERR(MC$)
 4240PROCVDWRD(FNITEM):ENDPROC
 4250
 4260DEFPROCprint
 4270SD$="":LOCALB$,C$,D$
 4280IFFNCUR=CHR$(&8A)PROCtab:GOTO4280
 4290IFFNCUR=""GOTO4370
 4300B$=FNITEM:IFB$=""ENDPROC
 4310IFLEFT$(B$,1)=CHR$(34)[jsrstring:]$
P%=MID$(B$,2,LEN(B$)-2):P%=P%+LEN(B$)-1:
GOTO4280
 4320IFB$="'"[jsr&FFE7:]GOTO4280
 4330IFB$=","[lda#32:jsrwr:]GOTO4280
 4340IFB$=";"GOTO4280
 4350PROCEXP(B$)
```

```
 4360[jsrprint:]GOTO4280
 4370IFB$=";"ENDPROC
 4380[jsr&FFE7:]ENDPROC
 4390
 4400DEFPROCsound
 4410SD$="":PROCSTWRD(FNITEM,&600):IFFNN
EXT<>",",PROCERR(MC$)
 4420PROCSTWRD(FNITEM,&602):IFFNNEXT<>",
",PROCERR(MC$)
 4430PROCSTWRD(FNITEM,&604):IFFNNEXT<>",
",PROCERR(MC$)
 4440PROCSTWRD(FNITEM,&606):[lda#7:ldx#0
:ldy#6:jsr&FFF1:]ENDPROC
 4450
 4460DEFPROCvdu
 4470LOCALB$,C$,B%:SD$=""
 4480C$=FNITEM:IFC$=""ENDPROC
 4490B$=FNNEXT:IFB$=";"PROCVDWRD(C$):GOT
O4480
 4500PROCVDBYT(C$):GOTO4480
 4510
 4520DEFPROCpoke
 4530SD$="":LOCALB$,C$:B$=MID$(A$,A%):C$
=MID$(B$,INSTR(B$,"=")+1):B$=LEFT$(B$,IN
STR(B$,"=")-1)
 4540B%=FNN(C$):IFB%<>-1[lda#(B%MOD256):
]GOTO4570
 4550B%=FNV(C$):IFB%<>-1[ldaB%:]GOTO4570
 4560PROCEXP(C$):[pla:tay:pla:tya:]
 4570B%=FNN(B$):IFB%<>-1[staB%:]ENDPROC
 4580B%=FNV(B$):IFB%<>-1[ldy#0:sta(B%),Y
:]ENDPROC
 4590[sta&8F:]PROCSTWRD(B$,&88):[lda&8F:
ldy#0:sta(&88),Y:]ENDPROC
 4600
 4610DEFPROCrepeat
 4620SD$="":R%(RE%)=P%:RE%=RE%+1:ENDPROC
 4630
 4640DEFPROCuntil
 4650SD$="":IFRE%=1PROCERR("UNTIL withou
```

```
t REPEAT")
 4660RE%=RE%-1:PROCEXP(MID$(A$,A%))
 4670[pla:tax:pla:txa:cmp#0:bneP%+5:jmpR
%(RE%):]ENDPROC
 4680
 4690DEFPROCfx
 4700A%=A%+2:SD$="":PROCSTBYT(FNITEM,&60
0):IFFNNEXT<>",PROCERR(MC$)
 4710PROCSTBYT(FNITEM,&601):IFFNNEXT<>",
"PROCERR(MC$)
 4720PROCSTBYT(FNITEM,&602):[lda&600:ldx
&601:ldy&602:jsr&FFF4:]ENDPROC
 4730
 4740DEFFNlinum(X%,Y%,Z%)
 4750LOCALL%:L%=0
 4760IFX%AND32THENL%=L%+128
 4770IF(X%AND16)=0THENL%=L%+64
 4780IF(X%AND4)=0THENL%=L%+16384
 4790IFY%AND32THENL%=L%+32
 4800IFY%AND16THENL%=L%+16
 4810IFY%AND8THENL%=L%+8
 4820IFY%AND4THENL%=L%+4
 4830IFY%AND2THENL%=L%+2
 4840IFY%AND1THENL%=L%+1
 4850IFZ%AND32THENL%=L%+8192
 4860IFZ%AND16THENL%=L%+4096
 4870IFZ%AND8THENL%=L%+2048
 4880IFZ%AND4THENL%=L%+1024
 4890IFZ%AND2THENL%=L%+512
 4900IFZ%AND1THENL%=L%+256
 4910=L%
 4920
 4930DEFPROCtosub(T%)
 4940SD$="":LOCALX%,Y%,Z%,LB%:IFFNNEXT<>
CHR$(141)PROCERR("Badly formed line numb
er")
 4950X%=ASC(FNNEXT):Y%=ASC(FNNEXT):Z%=AS
C(FNNEXT)
 4960LB%=FNlinum(X%,Y%,Z%)
 4970LL%(LC%,0)=LB%
```

```
 4980LL%(LC%,1)=P%
 4990LC%=LC%+1
 5000IFT%=TRUE THEN[jmpLB%:]ENDPROC ELSE
[jsrLB%:]ENDPROC
 5010
 5020DEFPROCfor
 5030SD$="":LOCALB$,C$,D$,B%,K%,D%
 5040B$=FNNEXT:IFB$>"Z"ORB$<"A"PROCERR("
Bad FOR variable")
 5050IFFNNEXT<>"="PROCERR("Missing =")
 5060C$=FNITEM:IFFNNEXT<>CHR$(&B8)PROCER
R("Missing TO")
 5070D$=FNITEM
 5080FO%(FR%,1)=ASC(B$)
 5090K%=&50+(ASC(B$)-65)*2
 5100PROCSTWRD(C$,K%)
 5110D%=FNN(D$):IFD%<>-1FO%(FR%,1)=FO%(F
R%,1)+D%*256:GOTO5130
 5120PROCEXP(D$)
 5130FO%(FR%,0)=P%
 5140FR%=FR%+1
 5150ENDPROC
 5160
 5170DEFPROCnext
 5180SD$="":LOCALB$,B%,C%,D%
 5190IFFR%=1PROCERR("NEXT without FOR")
 5200B$=FNNEXT
 5210IFB$>"Z"ORB$<"A"PROCERR("Bad NEXT v
ariable")
 5220FR%=FR%-1
 5230IFB$<>CHR$(FO%(FR%,1)AND&FF)PROCERR
("Wrong NEXT variable")
 5240B%=&50+(ASC(B$)-65)*2
 5250C%=FO%(FR%,0)
 5260IFFO%(FR%,1)<256[pla:sta&8C:pla:sta
&8D:cmpB%+1:bneP%+8:lda&8C:cmpB%:beqP%+1
7:lda&8D:pha:lda&8C:pha:incB%:bneP%+4:in
cB%+1:jmpC%:]ENDPROC
 5270D%=(FO%(FR%,1)AND&FFFF00)DIV256
 5280IFD%>255[lda#(D%DIV256):cmpB%+1:bne
```

```
P%+8:lda#(D%MOD256):cmpB%:beqP%+11:incB%
:bneP%+4:incB%+1:jmpC%:]ENDPROC
 5290[lda#D%:cmpB%:beqP%+7:incB%:jmpC%:]
ENDPROC
 5300
 5310DEFFNLINAD(Z%)
 5320LOCALT%:T%=1
 5330IFZ%=LI%(T%,0):=LI%(T%,1)
 5340T%=T%+1:IFT%<=L%GOTO5330
 5350PROCERR("Missing line "+STR$(Z%))
 5360
 5370DEFPROCsecondpass
 5380W%=&A00:$W%=CHR$(0)+"(C) 1982 Jerem
y Ruston"+CHR$(10)+CHR$(13)+CHR$(0)
 5390?&FD=W%MOD256:?&FE=W%DIV256
 5400LOCALT%,AD%:IFLC%=1ENDPROC
 5410FORT%=1TOLC%-1:P%=LL%(T%,1)+1
 5420AD%=FNLINAD(LL%(T%,0))
 5430PRINT~P%;"=";~AD%
 5440?P%=AD%MOD256:P%?1=AD%DIV256
 5450NEXTT%
 5460ENDPROC
 5470
 5480DEFPROCif
 5490SD$="":LOCALB$,C$,D$
 5500B$=FNITEN:IFFNNEXT<>CHR$(&8C)PROCER
R("Missing THEN")
 5510PROCEXP(B$)
 5520C$=FNITEN
 5530IFFNNEXT=CHR$(&8B)D$=FNITEN ELSED$=
""
 5540LB%=LI%(L%+1,0)
 5550[pla:tax:pla:txa:cmp#0:bneP%+5:]
 5560F%=P%+1
 5570[jmp32768:]
 5580PROCstatement(C$)
 5590IFD$=""LL%(LC%,0)=LB%:LL%(LC%,1)=F%
-1:LC%=LC%+1:ENDPROC
 5600LL%(LC%,0)=LB%:LL%(LC%,1)=P%:LC%=LC
%+1
```

```
5610[jmpLB%:]
5620?F%=P%MOD256:F%?1=P%DIV256
5630PROCstatement(D$)
5640ENDPROC
5650
5660DEFPROCinkey
5670LOCALB$,C$,C%
5680B$="":IFFNNEXT<>"("PROCERR("Missing
( after INKEY")
5690C%=0
5700C$=FNNEXT:IFC$=""PROCERR("Missing )
after INKEY")
5710IFC$=")"ANDC%=0GOTO5750
5720B$=B$+C$:IFC$=")"C%=C%+1
5730IFC$="("C%=C%-1
5740GOTO5700
5750[lda#124:jsr&FFF4:]C%=FNN(B$):IFC%<
>-1[ldx#(C%MOD256):ldy#(C%DIV256):lda#12
9:jsr&FFF4:tya:pha:txa:pha:]ENDPROC
5760C%=FNV(B$):IFC%<>-1[ldxC%:ldyC%+1:l
da#129:jsr&FFF4:tya:pha:txa:pha:]ENDPROC
5770PROCEXP(B$):pla:tax:pla:tay:lda#129
:jsr&FFF4:tya:pha:txa:pha:]ENDPROC
5780
5790DEFPROCrnd
5800LOCALB$,C$,C%
5810B$="":IFFNNEXT<>"("PROCERR("Missing
( after RND")
5820C%=0
5830C$=FNNEXT:IFC$=""PROCERR("Missing )
after RND")
5840IFC$=")"ANDC%=0GOTO5880
5850B$=B$+C$:IFC$=")"C%=C%+1
5860IFC$="("C%=C%-1
5870GOTO5830
5880PROCEXP(B$):[jsrrnd:]ENDPROC
5890
5900DEFPROCtab
5910LOCALB$,C$,C%:C%=0:A%=A%+1
5920C$=FNNEXT:IFC$=""PROCERR("Missing )
```

```
 after TAB")
 5930IFC$=")"ANDC%=0GOTO5970
 5940B$=B$+C$:IFC$=")"C%=C%+1
 5950IFC$="("C%=C%-1
 5960GOTO5920
 5970IFINSTR(B$,",")=0PROCERR("Wrong arg
uments for TAB")
 5980PROCstatement(CHR$(&EF)+"31,"+B$):E
NDPROC
 5990
 6000DEFPROCadval
 6010LOCALB$,C$,C%
 6020B$="":IFFNNEXT<>"("PROCERR("Missing
 ( after ADVAL")
 6030C%=0
 6040C$=FNNEXT:IFC$=""PROCERR("Missing )
 after ADVAL")
 6050IFC$=")"ANDC%=0GOTO6090
 6060B$=B$+C$:IFC$=")"C%=C%+1
 6070IFC$="("C%=C%-1
 6080GOTO6040
 6090C%=FNN(B$):IFC%<>-1[ldx#(C%MOD256):
ldy#(C%DIV256):lda#128:jsr&FFF4:tya:pha:
txa:pha:]ENDPROC
 6100C%=FNV(B$):IFC%<>-1[ldxC%:ldyC%+1:l
da#128:jsr&FFF4:tya:pha:txa:pha:]ENDPROC
 6110PROCEXP(B$):[pla:tax:pla:tay:lda#12
8:jsr&FFF4:tya:pha:txa:pha:]ENDPROC
 6120
 6130DEFPROCcompileF
 6140ONERRORGOTO6490
 6150LOCALwr,QQ%,A%,B%,C%,D%,T%
 6160wr=&FFEE:QQ%=OPENIN(FI$):L%=0:IFQQ%
=0PROCERR("No such file")
 6170A%=BGET#QQ%:B%=BGET#QQ%
 6180REPEAT
 6190C%=BGET#QQ%:D%=BGET#QQ%
 6200L%=L%+1
 6210FORT%=1TOD%-4:U%=BGET#QQ%:NEXT
 6220A%=BGET#QQ%:B%=BGET#QQ%
```

```
 6230UNTILB%=&FF
 6240DIMLI%(L%,1)
 6250CLOSE#0:QQ%=OPENIN(FI$):L%=0
 6260A%=BGET#QQ%:B%=BGET#QQ%
 6270REPEAT
 6280C%=BGET#QQ%:D%=BGET#QQ%
 6290L%=L%+1
 6300LI%(L%,0)=C%+B%*256
 6310FORT%=1TOD%-4:U%=BGET#QQ%:NEXT
 6320A%=BGET#QQ%:B%=BGET#QQ%
 6330UNTILB%=&FF
 6340CLOSE#0:QQ%=OPENIN(FI$)
 6350L%=0
 6360A%=BGET#QQ%:B%=BGET#QQ%
 6370REPEAT
 6380C%=BGET#QQ%:D%=BGET#QQ%
 6390L%=L%+1
 6400LI%(L%,1)=P%
 6410G$=""
 6420FORT%=1TOD%-4:G$=G$+CHR$(BGET#QQ%):
NEXT
 6430PROCoutput(G$)
 6440PROCstatement(G$)
 6450A%=BGET#QQ%:B%=BGET#QQ%
 6460UNTILB%=&FF
 6470CLOSE#0:[rts:]ENDPROC
 6480
 6490CLOSE#0:REPORT:PRINT" at line ";ERL
:END
 6500
 6510DEFPROCsave
 6520REPEAT
 6530INPUT'"Enter the filename : "A$
 6540UNTILLEN(A$)<=7
 6550DIME%40
 6560$E%="SAVE "+A$+" "+STR$~(M%)+"+"+ST
R$~(K%-M%)+" "+STR$~(N%)
 6570PRINT"*";$E%
 6580X%=E%MOD256:Y%=E%DIV256:CALL&FFF7
 6590PRINT"Successful save"
```

```
 6600ENDPROC
 6610
 6620DEFPROCSPRITE(A$)
 6630LOCALA%,B%,X%:A%=1
 6640B%=VAL(FNITEM):IFB%>70RB%<0PROCERR(
"Bad sprite")
 6650B%=B%*8+sp
 6660IFFNNEXT<>",","PROCERR(MC$)
 6670FORX%=0TO7
 6680B%?X%=EVAL(FNITEM)
 6690IFX%<>7IFFNNEXT<>",","PROCERR(MC$)
 6700NEXTX%
 6710ENDPROC
 6720
 6730DEFPROCDRAW(A$)
 6740LOCALA%,B%,X%,Y%,B$:A%=1
 6750B%=FNN(FNITEM):IFB%<00RB%>7PROCERR(
"Bad sprite")
 6760IFFNNEXT<>",","PROCERR(MC$)
 6770Y%=B%*8+sp
 6780B$=FNITEM:B%=FNN(B$):IFB%<>-1[ldx#(
B%MOD256):]GOTO6820
 6790B%=FNV(B$):IFB%<>-1[ldxB%:]GOTO6820
 6800PROCEXP(B$)
 6810[pla:tax:pla:]
 6820IFFNNEXT<>",","PROCERR(MC$)
 6830B$=FNITEM:B%=FNN(B$):IFB%<>-1[ldy#(
B%MOD256):]GOTO6870
 6840B%=FNV(B$):IFB%<>-1[ldyB%:]GOTO6870
 6850PROCEXP(B$)
 6860[pla:tay:pla:]
 6870[jmpP%+6:]!P%=Y%*256:P%=P%+3:[ldaP%
-2:sta&8D:ldaP%-6:sta&8E:jsrsprite:]
 6880ENDPROC
```

```
>LIST
   10
   20
   30
   40 REM Relocation program
   50 REM (c) 1982 Jeremy Ruston
   60
   70
   80
   90 REM ----------------------------
  100 REM Number of bytes in each
  110 REM instruction:
  120 DATA 3200022012100330
  130 DATA 2200022013000330
  140 DATA 3200222012103330
  150 DATA 2200022013000330
  160 DATA 1200022012103330
  170 DATA 2200022013000330
  180 DATA 1200022012103330
  190 DATA 2200022013000330
  200 DATA 0200222010103330
  210 DATA 2200222013300300
  220 DATA 2220222012103330
  230 DATA 2200222013103330
  240 DATA 2200222012103330
  250 DATA 2200022013000330
  260 DATA 2200222012103330
  270 DATA 2200022013000330
  280 REM ----------------------------
  290 REM User interface:
  300 RESTORE
  310 INPUT "Enter the filename"'"of the
 source program:"F$
  320 INPUT '"Enter the target address:"
A$'
  330 V%=EVAL(A$)
  340 INPUT "Enter the filename"'"of the
 object file:"O$
  350 REM ----------------------------
  360 REM Load file:
```

```
  370 DIM E% 18+LEN(F$)
  380 $(E%+19)=F$
  390 !E%=E%+19
  400 E%?6=1
  410 X%=E% MOD 256
  420 Y%=E% DIV 256
  430 A%=&FF
  440 CALL &FFDD
  450 REM ----------------------------
  460 REM Get file information:
  470 S%=E%!2
  480 G%=E%!6
  490 L%=E%!10
  500 REM ----------------------------
  510 REM Check validity:
  520 IF G%=0 THEN PRINT "Error - wrong
type of file":END
  530 REM ----------------------------
  540 REM Read instruction sizes:
  550 DIM I%255
  560 FOR T%=0 TO 15
  570 READ L$
  580 FOR G%=0 TO 15
  590 ?(I%+T%*16+G%)=VAL(MID$(L$,G%+1,1)
)
  600 NEXT G%,T%
  610 REM ----------------------------
  620 REM Relocate:
  630 P%=S%+64
  640 REPEAT
  650 T%=?(I%+(?P%))
  660 P%=P%+1
  670 IF T%=0 THEN PRINT "Error - unknow
n op-code encountered":END
  680 IF T%=2 THEN P%=P%+1
  690 IF T%=3 THEN PROCthree_byter
  700 UNTIL P%=S%+L%
  710 $E%="*SAVE "+O$+" "+STR$~(S%)+"+"+
STR$~(L%)+" "+STR$~(V%+64)+" "+STR$~(V%)
  720 PRINT $E%
```

```
730 X%=E% MOD 256
740 Y%=E% DIV 256
750 CALL &FFF7
760 END
770 REM ----------------------------
780 DEF PROCthree_byter
790 X%=?P%
800 Y%=P%?1
810 P%=P%+2
820 F%=X%+Y%*256
830 H%=(F%-S%)+V%
840 IF F%<S% OR F%>(S%+L%) THEN ENDPRO
C
850 IF F%<>S%+67 THEN P%?-2=H% MOD 256
:P%?-1=H% DIV 256:ENDPROC
860 P%?-2=H% MOD 256:P%?-1=H% DIV 256
870 REPEAT
880 P%=P%+1
890 UNTIL P%?-1=13
900 ENDPROC
910 REM ----------------------------
```